A-8/2193
8-27-07

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Appellant:	WISE	Examiner:	Mitchell, J.
Serial No.:	10/008,952	Group Art Unit:	2193
Filed:	December 6, 2001	Docket No.:	RA-5417 (USYS.030PA)
Title:	ARBITRARY AND EXPANDABLE HIGH-PRECISION DATATYPE AND METHOD OF PROCESSING		

CERTIFICATE UNDER 37 CFR 1.8: The undersigned hereby certifies that this correspondence and the papers, as described hereinabove, are being deposited in the United States Postal Service, as first class mail, in an envelope addressed to: Commissioner for Patents, Mail Stop Appeal Brief-Patents, P.O. Box 1450, Alexandria, VA 22313-1450, on September 19, 2006.

By: Kelly S. Waltigney
Kelly S. Waltigney

TRANSMITTAL

Mail Stop Appeal Brief-Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

We are submitting herewith the following:

☒ Transmittal Sheet containing executed Certificate of Deposit under 37 C.F.R. § 1.8.

☒ A Brief in Support of Appeal, pages 1-19, which includes an Appendix of Appealed Claims; an Appendix of Evidence with two attachments; and an Appendix of Related Proceedings (none).

☒ On page one, authorization is provided to charge deposit account 50-0996 (USYS.030PA) $500.00 and all required fees to enter this appeal brief.

☒ Return Postcard.

Please direct all correspondence to:
Charles A. Johnson, Esq.
Unisys Corporation
P.O. Box 64942 MS 4773
St. Paul, MN 55164

Respectfully submitted,

By: _____
Name: LeRoy D. Maunu
Reg. No.: 35,274

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Appellant:	WISE	Examiner:	Mitchell, J.
Serial No.:	10/008,952	Group Art Unit:	2193
Filed:	December 6, 2001	Docket No.:	RA-5417 (USYS.030PA)

Title: ARBITRARY AND EXPANDABLE HIGH-PRECISION DATATYPE AND METHOD OF PROCESSING

CERTIFICATE UNDER 37 CFR 1.8: The undersigned hereby certifies that this correspondence and the papers, as described hereinabove, are being deposited in the United States Postal Service, as first class mail, in an envelope addressed to: MAIL STOP APPEAL BRIEF-PATENTS, United States Patent and Trademark Office, P.O. Box 1450, Alexandria, VA 22313-1450, on September 19, 2006.

By: Kelly Waltigney
Kelly Waltigney

APPEAL BRIEF

Mail Stop Appeal Brief-Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

This is an Appeal Brief submitted pursuant to 37 C.F.R. §41.37 for the above-referenced patent application. Please charge **Deposit Account No. 50-0996 (USYS.032PA) in the amount of $500** for this brief in support of appeal as indicated in 37 C.F.R. § 41.20(b)(2). If necessary, authority is given to charge/credit deposit account 50-0996 (USYS.030PA) any additional fees/overages in support of this filing.

I. Real Party in Interest

The real party in interest is Unisys Corporation having a place of business at Township Line and Union Meeting Roads, Blue Bell, PA 19424.

II. Related Appeals and Interferences

Appellant is unaware of any related appeals, interferences or judicial proceedings.

III. Status of Claims

Claims 1-4 and 6-20 are rejected and are presented for appeal. Claims 5 and 21 are cancelled. The appealed claims are in the attached Appendix of Appealed Claims.

IV. Status of Amendments

There was no amendment filed after the final rejection of the claims.

V. Summary of Claimed Subject Matter

In one embodiment of the invention such as set forth in claim 1, the invention provides a computer-implemented method for processing numerical values in a computer program executable on a computer system. The method comprises encapsulating (Abstract, l. 3; p. 5, l. 2-14; Appendix A) in a large-integer datatype, large-integer data and associated operators. The large-integer data has runtime expandable precision and maximum precision that is limited only by system memory availability (FIG. 1, #104; p. 3, l. 24-26). Language-provided arithmetic, logical, and type conversion operators are overloaded with the large-integer operators that operate on large-integer variables in combination with other datatypes (p. 5, l. 2-14; Appendix A), and programmed usage of a variable of the large-integer datatype is equivalent to and interoperable with a variable of a system-defined integral datatype (p. 5, l. 2-14; Appendix A). A plurality of available storage nodes are established (FIG. 2, #206; p. 5, l. 28-29) and made available for allocation to large-integer data. A subset of the plurality of available storage nodes are allocated for a large-integer variable (FIG. 2, #210, #212; p. 5, l. 22-28). The subset is an allocated plurality of storage nodes. A numerical value is stored in the allocated plurality of storage nodes, and a linked list is formed of the allocated plurality of storage nodes (FIG. 1, #106-114, e.g.; p. 4, l. 11-12). A total number of available storage nodes available

for allocation to large-integer data is determined (FIG. 2, #218; p. 5, l. 25-26). Memory is allocated for a first number of available storage nodes, responsive to the total number being less than first threshold value, and the first number of available storage nodes is established (FIG. 2, #206, #214; p. 5, l. 24-29). Responsive to the total number being greater than a second threshold value, a second number of storage nodes is removed from the plurality of available storage nodes and memory for the second number of storage nodes is deallocated (FIG. 2, #224, #226; p. 5, l. 30 – p. 6, l. 3).

In another embodiment as set forth in claim 18, an apparatus is provided for processing numerical values in a computer program executable on a computer system (p. 3, l. 30-32; p. 5, l. 15-16; p. 6, l. 4-7; l. 14-17). The apparatus comprises means for encapsulating (Abstract, l. 3; p. 5, l. 2-14; Appendix A) in a large-integer datatype, large-integer data and associated operators, wherein the large-integer data has runtime expandable precision and maximum precision is limited only by system memory availability (FIG. 1, #104; p. 3, l. 24-26); means for overloading language-provided arithmetic, logical, and type conversion operators for integers with the large-integer datatype operators that operate on large-integer variables in combination with other datatypes (p. 5, l. 2-14; Appendix A), and programmed usage of a variable of the large-integer datatype is equivalent to and interoperable with a variable of a system-defined integral datatype (p. 5, l. 2-14; Appendix A); means for establishing a plurality of allocable storage nodes (FIG. 2, #206; p. 5, l. 28-29) available for allocation to large-integer data; means for allocating, for a large-integer variable, a subset of the plurality of allocable storage nodes (FIG. 2, #210, #212; p. 5, l. 22-28), the subset becoming an allocated plurality of storage nodes for the large-integer variable; and means for storing a numerical value in the allocated plurality of storage nodes and forming a linked list of the allocated plurality of storage nodes (FIG. 1, #106-114, e.g.; p. 4, l. 11-12).

VI. Grounds of Rejection

A. Claims 1-4, 6-12, 14, 17-18, and 20 stand rejected under 35 U.S.C. §103(a) as being unpatentable over "White" ("Reconfigurable, Retargetable Bignums" by White) in view of "Hardy" (U.S. Patent No. 5,640,496 to Hardy et al.), and further in view of "Carey" (US patent 6,078,994 to Carey)

B. Claim 13 stands rejected under 35 U.S.C. §103(a) as being unpatentable over the White-Hardy-Carey combination, further in view of "Burnikel" ("Fast Recursive Division" by Burnikel et al.)

C. Claim 15 stands rejected under 35 U.S.C. §103(a) as being unpatentable over the White-Hardy-Carey combination, further in view of "Esakov" ("Data Structures, an Advanced Approach Using C" by Esakov et al.)

D. Claims 16 and 19 stand rejected under 35 U.S.C. §103(a) as being unpatentable over the White-Hardy-Carey combination, further in view of "Anderson" (U.S. Patent No. 5,619,711 to Anderson)

VII. Argument

A. The rejection of claims 1-4, 6-12, 14, 17-18, and 20 should be reversed because the Examiner has not shown that the claims are unpatentable over the White-Hardy-Carey combination under 35 USC §103(a).

The Examiner has failed to establish a *prima facie* case of obviousness of claims 1-4, 6-12, 14, 17-18, and 20 over the White-Hardy-Carey combination because all the limitations are not shown to be suggested by the combination, a proper motivation for modifying White with teachings of Hardy and Carey has not been provided, and no showing is made that White could be modified with a reasonable likelihood of success.

Claims 1, 2, 3, 4, 6, 7, 8, 9, 14, 17, 18, 20

In claim 1, for example, the cited teachings of the White-Hardy-Carey combination neither teach nor suggest the claim limitations of allocating memory for a first number of available storage nodes, responsive to the total number being less than first threshold value, and establishing the first number of available storage

nodes; and removing a second number of storage nodes from the plurality of available storage nodes responsive to the total number being greater than a second threshold value, and deallocating memory for the second number of storage nodes.

In one Office Action the Examiner explained that "the teaching in Carey that is relied upon is simply the application of an optimal range of available nodes in a free list (minimum and maximum thresholds)." In a subsequent Office Action the Examiner explained that "one of ordinary skill in the art would ... not have replaced the existing allocation / deallocation means taught in Hardy .. with those of Carey, thus creating a free list manager with uses conventional allocation / deallocation from / to memory ... to maintain a number of free nodes between lower and upper thresholds..."

The cited portions of Carey neither teach nor suggest the claim limitations, and it is unclear from these explanations what teachings from Carey are proposed for making the combination.

Looking to how Carey accomplishes making available an optimal range of available nodes in a free list, it can be seen that the approach taught by Carey does not suggest or reasonably correspond to the claim limitations. Carey, in order to achieve the desired number of available free pages, does not allocate and deallocate memory as claimed. Rather, Carey teaches returning pages that are used to a list of free page buffers. Whether used or unused, the memory associated with Carey's free pages is always allocated to the application. There are no apparent calls to the system to either allocate additional memory if there are insufficient free pages or to deallocate memory for free pages if there are too many free pages. Thus, there is no allocation and deallocation of memory as claimed.

Carey teaches a shared cache system in which data retrieved from mass storage is stored in a page buffer taken from a free buffer list for access by users-sessions (col. 5, l. 1-4; 54-56). "To minimize processing delays which can occur when the free list becomes empty, the paging manager 15 maintains a counter of the number of entries on the free list." (col. 7, l. 40-42). When a minimum threshold is met, a collecting operation frees additional page buffers (col. 7, l. 43-46). The page buffers that are put back to the free list are taken from the page buffers that are

in use (col. 7, l. 64 – col. 8, l. 43). Thus, Carey reclaims page buffers that are in use, and Carey's approach would render the White-Hardy combination inoperable. If, in applying Carey's approach to the White-Hardy combination, memory used to store a bignum in the White-Hardy combination was collected while a program was using that bignum, the program's operation would be corrupted.

As explained in the MPEP §2143.01 (V) "If [the] proposed modification would render the prior art invention being modified unsatisfactory for its intended purpose, then there is no suggestion or motivation to make the proposed modification." *In re Gordon*, 733 F.2d 900, 221 USPQ 1125 (Fed. Cir. 1984). Therefore, the asserted modification to Hardy is improper.

Furthermore, neither Carey nor Hardy suggest the specific claim limitations of allocating and deallocating memory according to the threshold values. Carey teaches managing a list of free page buffers. Carey's allocation of memory for caching persistent data takes place at startup (col. 7, l. 60-63), which shows that Carey distinguishes between management of a free list and allocating and deallocating memory. The cited portion of Hardy simply teaches that "host system memory will be allocated in blocks of user-defined size..." (col. 8, l. 6-7). There is no suggestion of any deallocation of blocks, nor is there any suggestion of use of threshold values in the allocation and deallocation process.

The Office Action further fails to show that the White-Hardy-Carey combination suggests the limitations of storing a numerical value in the allocated plurality of storage nodes and forming a linked list of the allocated plurality of storage nodes. Hardy appears to store multiple pixel values in the linked list of pixel value nodes (col. 6, l 35-37). The linked list appears to accommodate a desired number of overlays of image values for a pixel location in an image (col. 5, l. 50-53). Thus, Hardy apparently stores one pixel value in each pixel value node in the linked list. This is not suggestive of the claimed storage of one value in a linked list of a plurality of nodes.

Thus, all the claim limitations are not shown to be suggested by the White-Hardy-Carey combination.

The alleged motivation for combining Hardy with White does not support a *prima facie* case of obviousness. The alleged motivation states that "it would have been obvious ... to use Hardy's methods of memory allocation/deallocation (col. 8, lines 4-27) with White's invention (pg. 176, col. 2, par. 3) to provide memory space for White's Bignums (pg. 177, par. 1 'Bignums are allocated in units of at least one 32-bit word') because one of ordinary skill in the art would have been motivated to provide an efficient memory management system (Hardy col. 8, line 4 'memory must be managed efficiently') to support White's disclosure of memory allocation (pg. 176, col. 2, par. 3)." This alleged motivation is improper because it is unsupported by evidence.

The alleged motivation draws the conclusion, without supporting evidence, that White's memory management is less efficient than that taught by Hardy. Furthermore, there is no evidence presented that provides reasons to replace White's approach to memory management with Hardy's approach. Without supporting evidence the alleged motivation is speculative and simply a hindsight-based reconstruction of the invention. Therefore, the alleged motivation to combine Hardy with White is insufficient to support *prima facie* obviousness.

Claims 2, 3, 4, 6, 7, 8, 9, 14, and 17 depend from claim 1, and a *prima facie case* of obviousness has not been established for at least the reasons set forth above. Independent claim 18 is directed to an apparatus and is drafted in mean-plus-function format. The claimed functions of claim 18 are not shown to be suggested by the White-Hardy-Carey combination for at least the reasons set forth above for claim 1, and claim 20 depends from 18 and is not shown to be unpatentable. Therefore, Applicant respectfully requests reversal of the rejection of claims 1, 2, 3, 4, 6, 7, 8, 9, 14, 17, 18, and 20.

Claim 10

The limitations of claim 10 are not shown to be suggested by White. Claim 10 includes limitations of overloading language-provided memory allocation and deallocation operators with large-integer operators that allocate and deallocate storage nodes. The Examiner cites White's "seamless interface between fixnums and bignums" as suggesting the limitations of overloading language-provided

memory allocation and deallocation operators with large-integer operators that allocate and deallocate storage nodes. However, even if one assumes that White's numerical operators are overloaded, it does not necessarily follow that memory allocation operators would also have to be overloaded to support allocation and deallocation of storage nodes as claimed.

In the Final Office Action, the Examiner explains that "providing a user invisible transition between fixnums and bignums would certainly imply to one of ordinary skill in the art that any action on a fixnum use the same function calls and / or operators as would be used on a bignum, and vice-versa[, otherwise] the distinction would be visible to the user, contradicting White's disclosure..." It is respectfully submitted that those skilled in the art will recognize that memory allocation and deallocation operations are not operations on a fixnum as the explanation states. Nor would they be operations a user would want to use in manipulating bignums. Contrary to the Examiner's reasons, to make manipulation of bignums seamless to the user, it would be undesirable to force the user to engage memory management operations. Therefore, the Office Action fails to show that the limitations of claim 10 are suggested by White. Therefore, Applicant respectfully requests reversal of the rejection of claim 10.

Claims 11, 12

The limitations of recursive operations as set forth in claim 11 are not shown be suggested by the White-Hardy-Carey combination along with the teachings of Knuth. The limitations of claim 11 include, responsive to a large-integer divide operation specifying an input dividend and divisor: identifying a set of most-significant bits of the dividend and a set of least-significant bits of the dividend; recursively performing a large-integer divide operation using the set of most-significant bits as the input dividend, and returning a quotient and a remainder; finding a lower-part dividend as a function of the remainder and the set of least-significant bits; recursively performing a large-integer divide operation using the lower-part dividend; and concurrently solving for the quotient and the remainder.

The Examiner admits that the White-Hardy-Carey combination does not suggest recursion, and the Examiner is incorrect in his asserted definition of

recursion. In the final Office Action the Examiner asserted that an "algorithm is recursive in that it repeatedly performs the same steps (d3-D6) on a smaller and smaller subset of the initial data ... until it reaches an ending or base condition..." It is respectfully submitted that those skilled in the art will recognize that the Examiner described an iterative approach, and recursion has a more specific definition. A function that calls itself is said to be recursive as demonstrated in pages 233-234 of Programming Language Concepts by Ghezzi and Jazayeri, 1982, which are attached in the Appendix of Evidence. The iterative approach shown in Knuth is not recursive. Therefore, the limitations of claim 11 are not shown to be suggested by the White-Hardy-Carey combination.

Claim 12 depends from claim 11 and the limitations are not shown to be suggested for at least the reasons set forth above. Therefore, Applicant respectfully requests reversal of the rejection of claims 11 and 12.

B. The rejection of claim 13 should be reversed because the Examiner has not shown that claim 13 is unpatentable under 35 USC §103(a)over the White-Hardy-Carey combination, further in view of Burnikel.

Claim 13 depends from claim 12, and the Examiner has not shown that all the limitations are suggested by the references and has not provided a proper motivation for modifying the teachings of the White-Hardy-Carey combination with teachings of Burnikel. Therefore, the rejection of claim 13 should be reversed.

C. The rejection of claim 15 should be reversed because the Examiner has not shown that claim 15 is unpatentable under 35 USC §103(a)over the White-Hardy-Carey combination, further in view of Esakov.

Claim 15 depends from claim 1, and the Examiner has not shown that all the limitations are suggested by the references and has not provided a proper motivation for modifying the teachings of the White-Hardy-Carey combination with teachings of Esakov. Therefore, the rejection of claim 15 should be reversed.

D. The rejection of claims 16 and 19 should be reversed because the Examiner has not shown that the claims are unpatentable under 35 USC §103(a) over the White-Hardy-Carey combination, further in view of Anderson.

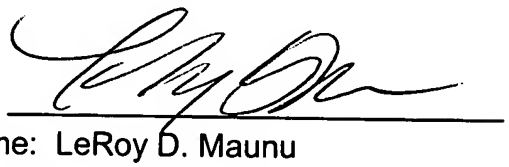
Claim 16 depends from claim 1, and claim 19 depends from claim 18. The Examiner has not shown that all the limitations are suggested by the references and has not provided a proper motivation for modifying the teachings of the White-Hardy-Carey combination with teachings of Anderson. Therefore, the rejection of claims 16 and 19 should be reversed.

VIII. Conclusion

In view of the above, Appellant submits that the rejections are improper, the claimed invention is patentable, and that the rejections of claims 1-4 and 6-20 should be reversed. Appellant respectfully requests reversal of the rejections as applied to the appealed claims and allowance of the entire application.

Respectfully submitted,

CRAWFORD MAUNU PLLC
1270 Northland Drive, Suite 390
Saint Paul, MN 55120
(651) 686-6633

By: 

Name: LeRoy D. Maunu
Reg. No.: 35,274

APPENDIX OF APPEALED CLAIMS FOR APPLICATION NO. 10/008,952

1. A computer-implemented method for processing numerical values in a computer program executable on a computer system, comprising:

encapsulating in a large-integer datatype, large-integer data and associated operators, wherein the large-integer data has runtime expandable precision and maximum precision is limited only by system memory availability;

overloading language-provided arithmetic, logical, and type conversion operators with the large-integer operators that operate on large-integer variables in combination with other datatypes, and programmed usage of a variable of the large-integer datatype is equivalent to and interoperable with a variable of a system-defined integral datatype;

establishing a plurality of available storage nodes available for allocation to large-integer data;

allocating a subset of the plurality of available storage nodes for a large-integer variable, the subset being an allocated plurality of storage nodes, and storing a numerical value in the allocated plurality of storage nodes and forming a linked list of the allocated plurality of storage nodes;

determining a total number of available storage nodes available for allocation to large-integer data;

allocating memory for a first number of available storage nodes, responsive to the total number being less than first threshold value, and establishing the first number of available storage nodes; and

removing from the plurality of available storage nodes, responsive to the total number being greater than a second threshold value, a second number of storage nodes and deallocating memory for the second number of storage nodes.

2. The method of claim 1, further comprising converting a character string into large-integer data in response to a constant definition statement.

3. The method of claim 2, further comprising converting large-integer data to and from a character string for input, output, and serialization.

4. The method of claim 1, further comprising:

converting input data from language-provided input functions to large-integer data; and

converting large-integer data to a format compatible with language-provided output functions.

6. The method of claim 1, further comprising allocating a selected number of bits for each storage node in response to a program-specified parameter.

7. The method of claim 1, further comprising dynamically allocating a number of storage nodes for storage of the numerical value as a function of a size of the numerical value.

8. The method of claim 7, further comprising storing in each node that is allocated to a large-integer variable, a subset of bit values that represent a numerical value.

9. The method of claim 8, further comprising:

maintaining a set of available storage nodes that are not allocated to any large-integer variable;

allocating a storage node from the set of available storage nodes to a large-integer variable while performing a large-integer operation that generates a numerical value and stores the numerical value in the variable, if a number of bit values required to represent the numerical value exceeds storage available in storage nodes allocated to the large-integer variable; and

returning to the set of available storage nodes a storage node allocated to a large-integer variable while performing a large-integer operation that generates a numerical value for storage in the variable, if a number of bit values required to

represent the numerical value is less than storage available in storage nodes allocated to the variable.

10. (original) The method of claim 9, further comprising overloading language-provided memory allocation and deallocation operators with large-integer operators that allocate and deallocate storage nodes.

11. The method of claim 1, further comprising, responsive to a large-integer divide operation specifying an input dividend and divisor:

identifying a set of most-significant bits of the dividend and a set of least-significant bits of the dividend;

recursively performing a large-integer divide operation using the set of most-significant bits as the input dividend, and returning a quotient and a remainder;

finding a lower-part dividend as a function of the remainder and the set of least-significant bits;

recursively performing a large-integer divide operation using the lower-part dividend; and

concurrently solving for the quotient and the remainder.

12. The method of claim 11, further comprising identifying an optimal set of most-significant bits of the dividend and a set of least-significant bits of the dividend as a function of a number of bits that represent the dividend and a number of bits that represent the divisor.

13. The method of claim 12, further comprising identifying an optimal set of most-significant bits of the dividend and a set of least-significant bits of the dividend as a function of one-half a difference between the number of bits that represent the dividend and the number of bits that represent the divisor.

14. The method of claim 1, further comprising emulating fixed-bit arithmetic on variables of the large-integer data type.

15. The method of claim 1, further comprising transferring data associated with temporary variables of the large-integer datatype by moving pointers to the data.

16. The method of claim 1, further comprising

encapsulating in a large-floating-point datatype, large-floating-point data and associated operators, wherein the large-floating-point data has runtime expandable precision and maximum precision is limited only by system memory availability; and

overloading language-provided arithmetic, logical, and type conversion operators for floating-point data with the large-floating-point datatype operators that operate on large-floating-point variables in combination with other datatypes, and programmed usage of a variable of the large-floating-point datatype is equivalent to and interoperable with a variable of a system-defined floating-point datatype.

17. The method of claim 1, further comprising

encapsulating in a large-rational datatype, large-rational data and associated operators, wherein the large-rational data has runtime expandable precision and maximum precision is limited only by system memory availability; and

overloading language-provided arithmetic, logical, and type conversion operators for rational data with the large-rational datatype operators that operate on large-rational variables in combination with other datatypes, and programmed usage of a variable of the large-rational datatype is equivalent to and interoperable with a variable of a system-defined rational datatype.

18. An apparatus for processing numerical values in a computer program executable on a computer system, comprising:

means for encapsulating in a large-integer datatype, large-integer data and associated operators, wherein the large-integer data has runtime expandable precision and maximum precision is limited only by system memory availability;

means for overloading language-provided arithmetic, logical, and type conversion operators for integers with the large-integer datatype operators that operate on large-integer variables in combination with other datatypes, and

programmed usage of a variable of the large-integer datatype is equivalent to and interoperable with a variable of a system-defined integral datatype;

means for establishing a plurality of allocable storage nodes available for allocation to large-integer data;

means for allocating, for a large-integer variable, a subset of the plurality of allocable storage nodes, the subset becoming an allocated plurality of storage nodes for the large-integer variable; and

means for storing a numerical value in the allocated plurality of storage nodes and forming a linked list of the allocated plurality of storage nodes.

19. The apparatus of claim 18, further comprising

means for encapsulating in a large-floating-point datatype, large-floating-point data and associated operators, wherein the large-floating-point data has runtime expandable precision and maximum precision is limited only by system memory availability; and

means for overloading language-provided arithmetic, logical, and type conversion operators for floating-point data with the large-floating-point datatype operators that operate on large-floating-point variables in combination with other datatypes, and programmed usage of a variable of the large-floating-point datatype is equivalent to and interoperable with a variable of a system-defined floating-point datatype.

20. The apparatus of claim 18, further comprising

means for encapsulating in a large-rational datatype, large-rational data and associated operators, wherein the large-rational data has runtime expandable precision and maximum precision is limited only by system memory availability; and

means for overloading language-provided arithmetic, logical, and type conversion operators for rational data with the large-rational datatype operators that operate on large-rational variables in combination with other datatypes, and programmed usage of a variable of the large-rational datatype is equivalent to and interoperable with a variable of a system-defined rational datatype.

APPENDIX OF EVIDENCE FOR APPLICATION NO. 10/008,952

The parameters following the "λ" and before the "." are called *bound variables*. When the lambda expression is applied, the occurrences of these variables in the expression following the "." are replaced by the arguments.

New functions may be created by combining other functions. The most common form of combining functions in mathematics is function composition. If a function F is defined as the composition of two functions G and H, written as

$F \equiv G \circ H$.

applying F is defined to be equivalent to applying H, and then applying G to the result. For example, if we define

to_the_fourth≡square∘square

then the value of *to_the_fourth (2)* is 16.

Function composition is an example of a *functional form*, also called a *combining form* or a *higher order function*. A functional form provides a method of combining functions; it is a function that takes other functions as parameters and yields a function as its result. Function composition is a functional form that takes two functions as parameters and yields the function that is equivalent to applying one and then the other as explained above.

8.2.2 Mathematical Functions Versus Programming Language Functions

Let us consider the differences between mathematical functions and those provided by conventional programming languages. The most important difference stems from the notion of a modifiable variable. The parameter of a mathematical function simply represents some value that is fixed at function application time; the function application results in another value. Data parameters in a programming language function, on the other hand, are names for memory cells; the function may use a name to change the contents in the cell. The function may also interact with the invoker of the function by changing other cells known to both of them. Because of these side effects, programming language functions, unlike mathematical functions, cannot always be combined hierarchically (like expressions).

Another significant difference is in the way functions are defined. In programming languages, a function is defined procedurally—the rule for mapping a value of the domain set to the range set is stated in terms of a number of steps that need to be "executed" in certain order specified by the control structure. Mathematical functions, on the other hand, are defined functionally—the mapping rule is defined in terms of applications of other functions. Mathematical functions are in this sense more hierarchical.

Many mathematical functions are defined *recursively*: that is, the definition of

the function contains an application of the function itself. For example, the standard mathematical definition of factorial is

n! = if n=0 then 1 else n* (n−1)!

The Fibonacci number sequence is defined as

F (n) = if n=0 or n=1 then 1 else F (n−1)+F (n−2).

As another example, we can readily formulate a (recursive) solution for the prime numbers problem of Section 8.1.1. The following predicate function determines whether a number is a prime:

prime (n) = if n=2 then true else p (n,n div 2)

where function p is:

p (n,i) = if (n mod i) = 0 then false
else if i=2 then true
else p (n,i−1)

Recursion is a powerful problem-solving technique. It is a natural and heavily used strategy, when programming with functions. However, as we have seen in Chapter 3, not all programming languages support recursive subprogram activations. When recursion is not permitted, the implementation of functions can become complicated and unnatural.

8.2.3 Functional (or Applicative) Languages

A functional programming language makes use of the mathematical properties of functions. In fact, the name "functional" (or "applicative") language is derived from the prominent role played by functions and function applications. A *functional (or applicative) language* has four components.

- A set of primitive functions.
- A set of functional forms.
- The *application* operation.
- A set of data objects.

The primitive functions are predefined by the language and may be applied to the functional forms are the mechanisms by which functions may be combined to form new functions. The application operation is the built-in mechanism for applying a function to its arguments and producing a value. The data objects are the allowed

APPENDIX OF RELATED PROCEEDINGS FOR APPLICATION NO. 10/008,952

Appellant is unaware of any related appeals, interferences or judicial proceedings.